Marti Cardoso

October 2019

1 Similarity Neural Network (SNN)

The similarity neural network (also called SNN) studied in this document, consists of a two-layer neural network, where the first layer computes the similarity between the input data and a set of prototypes, and the second gathers these results and predicts the output.

1.1 Framework

First, we are going to explain the main framework of this algorithm and secondly, it is explained how this network is trained. So, as said before, the SNN is a two-layer neural network where first it is computed the similarity between the input space and a set of prototypes and the second layer gathers all these similarities and predicts the output.

Given an input, called x, with n features (x_i is the feature i of input x), the goal is to predict some feature using the network. The first stage consists on computing the Gower similarity between the input and a set of m prototypes chosen in the learning stage (we will explain how to chose them in future sections), each prototype corresponds to a neuron in the hidden layer.

Then, once all the Gower similarities are computed, it is applied to these similarities an activation function. As the Gower similarities go from $[0, 1]$, we apply a sigmoid-like automorphism that given a value in the interval $[0, 1]$ it returns a value in $(0, 1)$. This activation function is called f_p and in future sections, it is explained in more detail.

At this point, we know the output of the hidden neurons. With these values, it is computed a linear model that generates the output (or outputs for the multinomial case) of the network.

Figure 3 shows a graphical representation of our network.

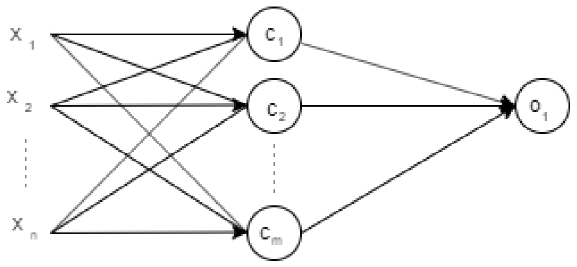

Figure 1: SNN framework.

We are going to see the network deeply.

1.1.1 First layer

The first layer computes the output of the hidden neurons by computing the similarities between the input and a set of prototypes selected in the learning stage. The similarity of Gower between two observations can be computed with the following formula.

$$s_g(x_i, x_j) = f_p\left(\frac{\sum_{k=1}^{n} s_k(x_{ik}, x_{jk}) \cdot w_k \cdot \delta_{ijk}}{\sum_{k=1}^{n} w_k \cdot \delta_{ijk}}\right)$$

where $s_k(x_{ik}, x_{jk})$ is the similarity between observation i and j for the feature k, $f_p(t)$ is the activation function, w_k is the weight of the feature and δ_{ijk} is the possibility of making the comparison between the two observations for the feature k (it will be 1 if it can be compared, 0 otherwise).

The output of the hidden neurons are:

$$h_i(x) = s_g(x, p_i) \qquad \text{for } i \text{ in } 1..M \tag{1}$$

where $h_i(x)$ is the output of the hidden neuron i and p_i is the prototype corresponding to this layer.

Activation function

The *activation* function f_p is any sigmoid-like automorphism (a monotonic bijection) in $[0, 1]$. In particular, the widespread *logistic* function can be used by adapting it to map the real interval

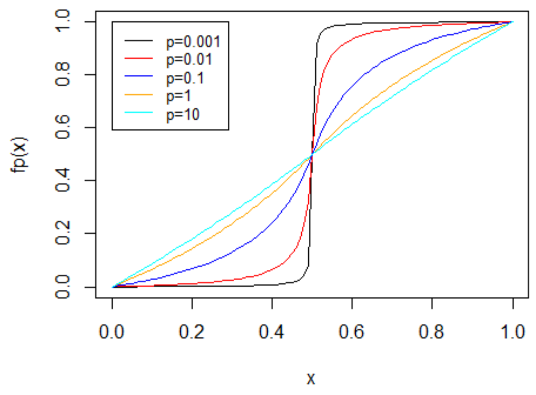

Figure 2: Activation function f_p for several p values.

$[0,1]$ on $(0,1)$. Computationally cheap families of sigmoidal functions can be especially designed to operate in the $[0,1]$ interval, such as $f_p(\cdot) = f(\cdot, p)$:

$$f(x,p) = \begin{cases} \frac{-p}{(x-0.5)-a(p)} - a(p) & \text{if } x \leq 0.5 \\ \frac{-p}{(x-0.5)+a(p)} + a(p) + 1 & \text{if } x \geq 0.5 \end{cases} \tag{2}$$

where $p > 0 \in \mathbb{R}$ is a parameter controlling the shape. This family of functions are all continuous bijections in $[0,1]$, fulfilling $\forall p \in \mathbb{R}+,\ f(0,k) = 0,\ f(1,k) = 1,\ \lim_{k\to\infty} f(x,p) = x$ and $f(x,0) = H(x-0.5)$, being H the Heaviside function. The expression $a(p)$ is solution of $a(p)^2 + \frac{a(p)}{2} - p = 0$, obtained by imposing the first two equalities.

Figure 2 shows a graphical representation of the behavior of this function depending on p.

1.1.2 Second layer

The second layer creates a linear model that receives the similarities computed in the previous layer and predicts the response variable.

The type of the response determines the activation function of the output neurons. The SNN can deal with three types of response:

- **Numerical**. In this case, we have a regression framework, and the activation function used is the identity.

$$\hat{y}(x) = \sum_{j=1}^{M} w_j \cdot h_j(x)$$

  where w_k are the weights of the linear model (these weights were found in the learning stage).

- **Binary**. This variable corresponds to logistic regression. The activation function is the sigmoid function.

$$\hat{y}(x) = \frac{1}{1 + e^{\left(\sum_{j=1}^{M} w_j \cdot h_j(x)\right)}}$$

- **Multinomial**. When the response variable is multinomial, it is used *softmax* as the activation function. It has an output neuron for each modality of the response.

$$\hat{y}_k(x) = \frac{e^{snn(x,k)}}{\sum_{k2=1}^{K} e^{snn(x,k2)}} \text{ where } snn(x,k) = \sum_{j=1}^{M} w_{jk} \cdot h_{jk}(x)$$

2 Learning stage

In this section, it is explained how the SNN is trained. First, we explain how to select the prototypes to be used as hidden neurons, then how to set the p parameter of the f_p function and finally how to train the linear model.

2.1 Prototype selection

The prototype selection is the first thing of the SNN to be learned. First, we need to decide the number of prototypes to use and secondly the selected observations to be prototypes.

2.1.1 Number of prototypes

The number of prototypes can be generated by four methods. Each one uses a different distribution to choose the number of prototypes. We should be able to define how are these distributions, so, for this reason, we introduce a hyper-parameter, called *hp*, that in most of the method means the expected proportion of observation that will be selected as prototypes. These are the four methods:

- **Constant**(C). This is the simplest method. The number of prototypes depends only on the size of the problem, and for the same problem, it will always generate the same value. In this case, the *hp* hyper-parameter means the proportion of observations that will be selected as prototypes.

$$M = \lfloor hp \cdot N \rfloor$$

- **Uniform** (U). This method uses the uniform distribution to set the number of prototypes. It generates a value between 1 and $hp \cdot N$ using the uniform distribution, so consecutive runs of the method will generate different values. In this case, the *hp* hyper-parameter means the maximum proportion of observations that could be selected as prototypes.

$$M = unif\left(1, hp \cdot N\right)$$

- **Binomial** (B). This method uses the binomial distribution to set the number of prototypes. In this distribution, the *hp* is used as the probability of success on each trial. So, as we have set the number of trials to be the number of observations, this distribution will have an expected value of $hp \cdot N$.

$$M = binomial\left(N, hp\right)$$

- **Poisson** (P). This method uses the Poisson distribution to set the number of prototypes. We set $\lambda = hp \cdot N$, so the expected value of the distribution is $hp \cdot N$.

$$M = poisson\left(\lambda = N \cdot hp\right)$$

2.1.2 Selection of prototypes

Once we know the number of prototypes, we have to select which observations are prototypes. We have defined two different methods to do this task:

- **PAM**. The first method consists of applying the clustering method PAM (Partitioning Around Medoids). Applying this clustering, we find groups of observations that are similar to each other and that have as a representative observation the medoid of each group. So, this first method consists of selecting the prototypes as the medoids found with the PAM algorithm. The benefit of this method is that it can select representative observations that are different from each other, but this clustering has a high computational cost.

- **Random**. The second method consists of selecting the prototypes randomly from all observations. It is the opposite of the previous method: it is very fast when selecting the prototypes, but as they are selected randomly, the observations can not be very representative and can be very similar to each other.

In future sections, it is analyzed if the use of PAM is useful for predicting tasks, or it gets similar results than using random, and thus, we can save computational time using the random method.

2.2 Choosing the value of p (f_p)

Once the prototypes are selected, we have to decide the value of p of the f_p function. We define four methods to set this value:

Constant

The first and simplest method is to set the p as a hyper-parameter that is predefined as $p = 0.1$, but the user can set it as he wants.

Optimization of p

At this point, the prototypes have been selected, so we can compute the output values of the hidden neurons before the activation function is applied (f_p). This method tries to optimize the value of p at the same time it does with the linear model. So, it does an iterative optimization procedure that first creates the linear model and secondly optimizes p for this model and iterates this procedure until some stopping criteria are reached.

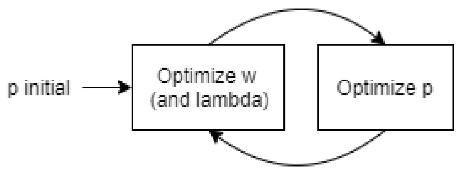

Figure 3: p optimization schema

First, we define an initial value of p that will be used to apply f_p to the similarities and then it is created the linear model that predicts the response variable. Once the model is created, we optimize the value of p for the found model (weights and lambda). It is used the BFGS method in order to do the optimization, and it is optimized the same objective function used in the linear model. Once we found the optimum value of p for the model, we apply again f_p and recompute the model, and it is started a new optimization iteration.

We stop the iterative procedure when one of the following criteria occurs:

- A maximum number of iterations.
- Change on p between two consecutive iterations is lower than a tolerance
- Change on the objective function between two consecutive iterations is lower than a tolerance.

The optimization of p and the creation of the model are very related, one uses the result of the other, so it could be needed several iterations in order to reach the optimum value.

The objective function to minimize depends on the type of the response variable: for the regression case, it is used the MSE and for the binomial and multinomial the cross-entropy. And for the initial value of p, we have defined two strategies: use a predefined initial value or try values between 0 and 1, and take the starting p as the one with better results.

For this method, we thought that we should use the **early stopping** method in order to avoid overfitting and do a regularization. This method consists of splitting the learning set into train and validation, apply the previously explained optimization to the train set, and also test the results with the validation set. Then, the value of p chosen is the one that performs better in the validation set.

k-fold cross-validation

As an alternative method to the previous one, it is implemented a k-fold cross-validation. It is tested several values of p between 0 and 1, and the one with the best validation error is selected as the best p.

Estimators

Finally, the last method consists of estimating the value of p by looking at the clustering results, and see how are the clusters, how mixed or disjoint are they,... These are the methods:

- **E1**: Mean of silhouette coefficients. This value goes from 0 to 1, where 0 means that the clusters are very disjoint (so the f_p function can be very discriminant) and 1 means that the clusters are very mixed (so the f_p function should be near-linear).

$$p = 1 - \frac{\sum_{j=1}^{N} s_i}{2N} \qquad s_i = \frac{a(i) - b(i)}{max(a(i), b(i))} \qquad (s_i \in [-1, 1])$$

  where $a(i)$ is the mean dissimilarity between i and all other points in the cluster to which i belongs and $b(i)$ is the minimum value between the mean distance to points of another cluster.

- **E2**. It is the same than the first method, but now $b(i)$ is the mean instead of the minimum.

- **E3**. This method is a coefficient based in the *Calinski-Harabasz* index:

$$p = \frac{\text{Mean distance between observations and its cluster medoid}}{\text{Mean distance between medoids}}$$

2.3 Linear models

Once we have the outputs of the hidden neurons, we have to train a linear model that predicts the response variable. Depends on the response variable it is fitted a type of linear model: for the regression case, it is fitted a linear model, for the binomial case we fit a logistic regression model and for the multinomial case we fit a multinomial log-linear model.

Also, if needed a regularization term can be added to these models (these λ will be trained with CV).

3 Annex

4 Derivatives

In this section, it is shown the derivatives used for the optimization of p (fp):

4.1 Regression

$$E(p) = \sum_{n=1}^{N} (y_i - \hat{y}(p, x_n))^2$$

$$\hat{y}(p, x) = \sum_{j=1}^{M} w_j f_p\left(s_g(x, p_j)\right) + w_0$$

where p_i means the prototype number i.

$$f_p(x) = \begin{cases} \frac{-p}{(x-0.5)-a(p)} - a(p) & \text{if } x \leq 0.5 \\ \frac{-p}{(x-0.5)+a(p)} + a(p) + 1 & \text{if } x \geq 0.5 \end{cases} \tag{3}$$

$$a(p) = -1/4 + \sqrt{0.5^4 + p}$$

Derivative (Regression)

$$\frac{\partial E(p)}{\partial p} = -2 \sum_{i=1}^{N} \left((t_i - \hat{y}(p, x_i)) \frac{\partial \hat{y}(p, x_i)}{\partial p} \right)$$

$$\frac{\partial \hat{y}(p, x)}{\partial p} = \sum_{j=1}^{M} w_j \frac{\partial f_p(s_g(x, c_j))}{\partial p}$$

$$\frac{\partial f_p(x)}{\partial p} = \begin{cases} \frac{-(x-0.5)+a(p)-p\frac{\partial a(p)}{\partial p}}{(x-0.5-a(p))^2} - \frac{\partial a(p)}{\partial p} & \text{if } x \leq 0.5 \\ \frac{-(x-0.5)-a(p)+p\frac{\partial a(p)}{\partial p}}{(x-0.5+a(p))^2} + \frac{\partial a(p)}{\partial p} & \text{if } x \geq 0.5 \end{cases} \tag{4}$$

$$\frac{\partial a(p)}{\partial p} = \frac{1}{2\sqrt{0.5^4 + p}}$$

4.1.1 Multinomial

$$E(p) = -\sum_{n=1}^{N}\left(\sum_{k=1}^{K} y_{nk} \cdot ln(\hat{y}_k(p, x_n))\right)$$

where y_{ij} is 1 when the row i is of the k modality, 0 otherwise.

$$\hat{y}_k(p, x) = \frac{e^{snn(x,p,k)}}{\sum\limits_{t=1}^{K} e^{snn(x,p,k)}}$$

$$snn(x, p, 1) = 0 \qquad\qquad snn(x, p, k) = \sum_{j=1}^{M} w_{jk} f_p\left(s_g(x, c_j)\right) + w_{0k}$$

Derivative (Multinomial)

Now, we compute the derivative of the E function:

$$\frac{\partial E}{\partial p} = -\sum_{n=1}^{N}\sum_{k=1}^{K}\left(y_{nk}\frac{1}{\hat{y_k}(p, x_n)}\frac{\partial \hat{y_k}(p, x_n)}{\partial p}\right)$$

$$\frac{\partial \hat{y_k}(p, x)}{\partial p} = \frac{e^{snn(x,p,k)}\frac{\partial snn(x,p,k)}{\partial p}\sum\limits_{t=1}^{K} e^{nnet(x,p,t)} - e^{snn(x,p,k)}\sum\limits_{t=1}^{K}\left(e^{snn(x,p,t)}\frac{\partial snn(x,p,t)}{\partial p}\right)}{\left(\sum\limits_{t=1}^{K} e^{nnet(x,p,t)}\right)^2}$$

$$\frac{\partial snn(x, p, 1)}{\partial p} = 0 \qquad\qquad \frac{\partial snn(x, p, k)}{\partial p} = \sum_{j=1}^{M} w_{jk}\frac{\partial f_p}{\partial p}\left(s_g(x, c_j)\right)$$

4.1.2 Binomial

$$E(p) = -\sum_{n=1}^{N} \big(y_n ln(\hat{y}(p, x_n)) + (1 - y_n) \cdot ln(1 - \hat{y}(p, x_n) \big)$$

$$\hat{y}_k(p, x) = sigmoid\,\big(snn(p, x)\big) \qquad\qquad sigmoid(x) = \frac{1}{1 + e^{-x}}$$

$$snn(p, x) = \sum_{j=1}^{M} w_j f_p\,\big(s_g(x, p_j)\big) + w_0$$

Now, we compute the derivative of the E function:

$$\frac{\partial E}{\partial p} = -\sum_{n=1}^{N} \left(y_n \frac{1}{\hat{y}(p, x_n)} \frac{\partial \hat{y}(p, x_n)}{\partial p} + (1 - y_n) \frac{1}{1 - \hat{y}(p, x_n)} \frac{-\partial \hat{y}(p, x_n)}{\partial p} \right)$$

$$\frac{\partial \hat{y}(p, x)}{\partial p} = \frac{\partial sigmoid\,\big(snn(p, x)\big)}{\partial p} \frac{\partial snn(p, x)}{\partial p}$$

$$\frac{\partial snn(x, p)}{\partial p} = \sum_{j=1}^{M} w_j \frac{\partial f_p}{\partial p}\,\big(s_g(x, c_j)\big)$$